C3D2-Themenabend

Erlang/OTP

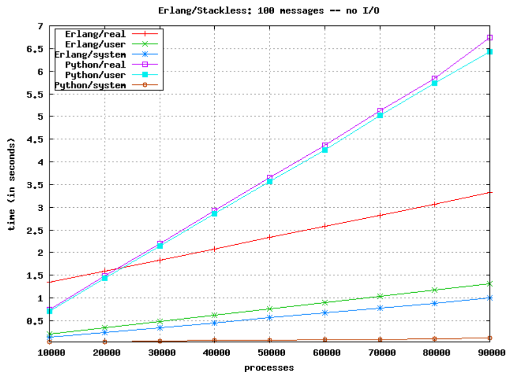
XX.12.2008

xmpp:astro@spaceboyz.net

Geschichte

Agner Krarup Erlang (1878 – 1929)
Ericsson Language
1986, 1998

Konzept



http://muharem.wordpress.com/2007/07/31/erlang-vs-stackless-python-a-first-benchmark/

Hilfe

```
$ erl -man erlang
```

www.erlang.org
www.trapexit.org

Start

```
$ erl
Erlang (BEAM) emulator version 5.6.4
 [source] [async-threads:0] [kernel-
poll:false]

Eshell V5.6.4  (abort with ^G)
1>
```

Start in emacs

```
M-x erlang-shell
M-x erlang-mode C-c C-k
```

Alles hat ein Ergebnis

Deja-vu

Terme

```
foo.
foo, bar.
foo, bar, baz.
```

Alternativen

```
foo; bar
foo; bar; baz
```

Variablen

beginnen mit Großbuchstaben, sind ungebunden oder gebunden

Matching

```
1> T = 3.
3
2> T = 3.
3
3> T = 4.
** exception error: no match of right hand
 side value 4
```

Ungenutzte Variablen

beginnen mit _

```
1> _ = 23.
23
2> _ = 42.
42
```

Module

```
-module(worldctl).
```

Funktionen

```
f() ->
      23.
```

Alternative Funktionen

```
f(foo) ->
      23;
f(bar) ->
      42.
```

Funktionen aufrufen

Im Modul:

```
f()
```

Aus anderem Modul:

```
worldctl:f()
```

Funktionen referenzieren

```
fun f/0
fun worldctl:f/2
```

Keine Schleifen

recursion: see recursion

Zählschleife

```
countdown(0) -> ok;
countdown(N) -> countdown(N - 1).
```

not tail-recursive

```
f() ->
      1 + f().
```

Anonyme Funktionen

```
fun() ->
      23
end.
```

Anonyme Funktionen mit Alternativen

```
fun(foo) ->
      23;
   (bar) ->
      42
end.
```

Anonyme rekursive Funktionen?

Y-Combinator

```
F = fun(F) ->
      F(F)
end,
F(F).
```

case

```
case Term of
  Pattern1 ->
      Action1;
  Pattern2 ->
      Action2;
  Pattern3 when Guard3 ->
      Action3;
  Pattern4 when Guard4 ->
      Action4
end.
```

if

```
if
  Guard1 ->
      Action1;
  Guard2 ->
      Action2;
  true ->
      DefaultAction
end.
```

Guards

```
... when V =/= 42 ->
... when V =/= 42, V =/= 23 ->
... when V =:= 42; V =:= 23 ->
```

Nur BIFs!

Built-in Functions (BIFs)

```
abs(Number) -> int() | float()
apply(Fun, Args) -> term() | empty()
apply(Module, Function, Args) -> term() |
 empty()
...
```

Exceptions

```
2> catch 10 = 5 + 6.
{'EXIT',{{badmatch,11},
         [{erl_eval,expr,3}]}}
```

catch ... -> Result | {'EXIT', {What, Where}}

Exceptions generieren

```
1> catch erlang:error(foo).
{'EXIT',{foo,[{erl_eval,do_apply,5},
              {erl_eval,expr,5},
              {shell,exprs,6},
              {shell,eval_exprs,6},
              {shell,eval_loop,3}]}}
2> catch exit(foo).
{'EXIT',foo}
3> catch throw(foo).
foo
```

Datentypen

Atome

```
foo, bar, baz,
'or anything with single quotes'.
```

Nummern

Integers:

`23, -42, 235.`

Floats:

`3.141592653589793.`

Tupel

```
{foo, 23, 42}.
```

Listen

```
[],
[23, 42, 235],
[23 | [42 | [235 | []]]].
```

Key-Value Lists

```
1> process_info(self()).
[{current_function,{erl_eval,do_apply,5}},
 {initial_call,{erlang,apply,2}},
 {status,running},
 {message_queue_len,0},
 {messages,[]},
 {links,[<0.25.0>]},
 {dictionary,[]},
 {trap_exit,false},
 {error_handler,error_handler},
 {priority,normal},
 {group_leader,<0.24.0>},
 {total_heap_size,1974},
 {heap_size,987},
 {stack_size,24},
 {reductions,1299},
 {garbage_collection,[{fullsweep_after,65535},{minor_gcs,5}]},
 {suspending,[]}]
```

Strings

```
1> [$H, $e, $l, $l, $o, 32, 87, 111, 114,
 108, 100].
"Hello World"
```

lists:map/2

```
lists:map(fun(N) -> N * 2 end,
          Numbers).
```

lists:filter/2

```
lists:filter(fun(N) -> N rem 2 == 0 end,
             Numbers).
```

List Comprehensions (1)

```
[N * 2
 || N <- Numbers,
    N rem 2 == 0].
```

List Comprehensions (2)

```
[{N * 2, M}
 || N <- lists:seq(1,10),
    M <- lists:seq(1,10),
    M rem 2 == 0].
```

lists:foldl/2

```
lists:foldl(fun(N, S) -> N + S end,
            0, Numbers).
```

References

```
1> make_ref().
#Ref<0.0.0.53>
```

Weitere Datentypen

PIDs
Nodes
Ports

Records

Strukturierte Tupel (vgl. struct in C)

Record definieren

```
-record(person, {name, age = 23}).
```

Neuer Record

```
#person{}
 = {person, undefined, 23},
#person{name = "Hans"}
 = {person, "Hans", 23},
#person{name = "Frank", age = 42}
 = {person, "Frank", 42}.
```

Auf Felder zugreifen

```
User#person.age,
User#person.name.
```

Pattern Matching mit Records

```
person_name(#person{name = Name}) -> Name;
person_name(_) -> "Anonym".
```

Records durch Kopie ändern

```
let_age(#person{age = Age} = P) ->
 P#person{age = Age + 1}.
```

Binaries

```
read_header(<<HData:12/binary, RData/binary>>) ->
    <<ID:16/unsigned, QR:1, OpCode:4, AA:1, TC:1,
     RD:1, RA:1, Z:3, RCode:4, QDCount:16/unsigned,
     ANCount:16/unsigned, NSCount:16/unsigned,
     ARCount:16/unsigned>> = HData,
    {#dns_header{id=ID, qr=QR, opcode=OpCode, aa=AA, tc=TC,
                rd=RD, ra=RA, z=Z, rcode=RCode, qdcount=QDCount,
                ancount=ANCount, nscount=NSCount, arcount=ARCount},
     RData}.
```

Macros

```
-define(FOO, 23).
?FOO.
```

Macros & Parameter

```
-define(SECOND(List), lists:nth(2, List)).
?SECOND([23, 42]).
```

Konkurrente Programmierung

Neuer Prozess

```
spawn(fun() -> ... end).
```

Nachrichten senden

```
Pid ! Term.
```

Nachrichten empfangen

```
receive
 Pattern1 -> Action1;
 Pattern2 when Guard2 -> Action2
end.
```

Nachrichten empfangen mit Timeout

```
receive
 Pattern1 -> Action1;
 Pattern2 when Guard2 -> Action2
 after Timeout -> TimeoutAction
end.
```

Linking

```
link(Pid),
spawn_link(fun() -> ... end).
```

trap_exit

```
process_flag(trap_exit, true),
receive
      {'EXIT', From, Reason} -> ...
end.
```

Registering PIDs

```
register(queue, Pid),
queue ! {enqueue, Item}.
```

Sockets

Active Sockets (1)

```
{active, false};
{active, once};
{active, true}.
```

Active Sockets (2)

```
receive ->
      {tcp, Sock, Bin} -> Action1;
      {tcp_closed, Sock} -> Action2;
      {tcp_error, Sock, Reason} -> Action3
end.
```

OTP

Open Telecom Platform — erweiterte Standardbibliothek

gen_server

Was erwartet man von einem Server?

Key-Value Storage Server (1)

```
-module(kv).
-export([loop/1]).

loop(KV) ->
  receive
    {From, get, Key} ->
        case lists:keysearch(Key, 1, KV) of
          {value, {Key, Value}} -> From ! {value, Value};
          false -> From ! not_found
        end,
        NewKV = KV;
    {From, set, Key, Value} ->
        NewKV = [{Key, Value} | lists:keydelete(Key, 1, KV)],
        From ! set
  end,
  ?MODULE:loop(NewKV).
```

Key-Value Storage Server (2)

```
-export([start_link/0, get/2, set/3]).

start_link() ->
  spawn_link(fun() -> loop([]) end).

get(Pid, Key) ->
  Pid ! {self(), get, Key},
  receive
      {value, Value} -> Value;
        not_found -> exit(not_found)
  end.

set(Pid, Key, Value) ->
  Pid ! {self(), set, Key, Value},
  receive
      set -> ok
  end.
```

gen_server API (1)

```
gen_server:start_link(Module, Args,
                      Options)
 -> {ok, Pid} | ignore | {error, Error}

gen_server:start_link(ServerName, Module,
                      Args, Options)
 -> {ok, Pid} | ignore | {error, Error}
```

gen_server Callbacks (1)

```
init(Args)
 -> {ok, State}

terminate(Reason, State)

code_change(OldVsn, State, Extra)
 -> {ok, NewState}
```

gen_server Callbacks (2)

```
handle_call(Request, From, State)
 -> {reply, Reply, NewState}
  | {noreply, NewState}
  | {stop, Reason, Reply, NewState}

handle_cast(Request, State)
 -> {noreply, NewState}
  | {stop, Reason, NewState}

handle_info(Info, State)
 -> {noreply, NewState}
  | {stop, Reason, NewState}
```

gen_server API (2)

```
call(Name | Pid, Request)
 -> Reply
call(Name | Pid, Request, Timeout)
 -> Reply

cast(Name | Pid, Request)
```

Mnesia

- Relational
- Transactional
- In-Memory
- Erlang-only
- Distributed

Mnesia: Creating a RAM-based table

```
-record(person, {name, age}).

mnesia:create_table(person,
                    [{attributes,
                      record_info(fields, person)}]).
```

Mnesia: Operations

```
read(Tab, Key)
 -> [Records]

write(Record)
write(Tab, Record, LockKind)

delete(Tab, Key, LockKind)
delete_object(Record)
delete_object(Tab, Record, LockKind)
```

Mnesia: Transactions

```
mnesia:transaction(fun() -> ... end)
 -> {atomic, ResultOfFun}
  | {aborted, Reason}
```

Mnesia: Locking (1)

```
F = fun() ->
      Values = mnesia:read(table, Key),
      NewValues = waste_cpu(Values),
      [mnesia:write(Value)
       || Value <- NewValues]
    end,
mnesia:transaction(F).
```

Mnesia: Locking (2)

```
F = fun() ->
      mnesia:write_lock_table(table),
      Values = mnesia:read(table, Key),
      NewValues = waste_cpu(Values),
      [mnesia:write(Value)
       || Value <- NewValues]
    end,
mnesia:transaction(F).
```

Mnesia: Dirty Operations

```
dirty_read(Tab, Key)

dirty_write(Record)

dirty_delete(Tab, Key)
dirty_delete_object(Record)
```

Mnesia: Match Expressions (1)

```
mnesia:select(channel_user,
              [{#channel_user{client = Nick,
                  channel = '$1'},
               [], ['$1']}])
```

Mnesia: Match Expressions (2)

```
mnesia:select(person,
              [{#person{age = '$1',
                         _ = '_'},
               [{orelse,
                 {'<', '$1', 18},
                 {'>', '$1', 67}}],
               ['$_']}])
```